Tutorial de uso de Android Asynchronous HTTP Client (LOOPJ)

Programación en Internet

Angel Manuel Gamaza Domínguez

José Miguel Otte Sainz-Aguirre

Grado en Ingeniería Informática

20 de septiembre de 2016

Índice

Índice de figuras

1. ¿Qué es LOOPJ y para qué sirve?

LOOPJ es una biblioteca externa de Android cuya función principal es la manipulación de llamadas HTTP de forma asíncrona. Dado que la biblioteca que proporciona actualmente Android está obsoleta, consideramos que esta es una buena opción a usar.

2. Como añadirla la biblioteca al IDE

- Abrimos el fichero **build-gradle** dentro de la sección Gradle Scripts en Android Studio.

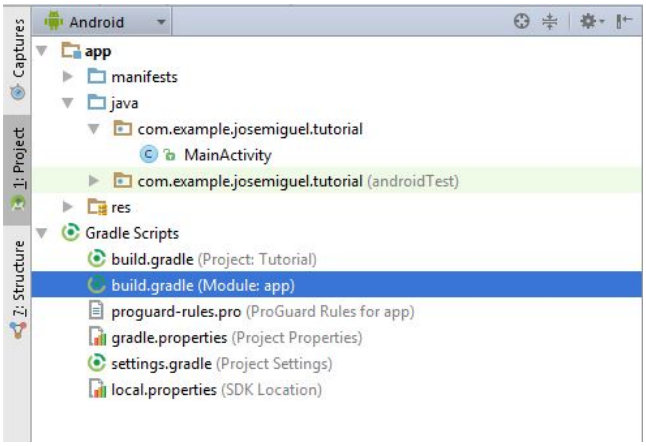

Figura 1: Ubicación del build-gradle.

- En "**Dependencies**" debemos incluir la siguiente línea.

```
dependencies {
  compile 'com.loopj.android:android-async-http:1.4.9'
}
```

Una vez incluida, debemos volver a sincronizar el Gradle. Para ello, bastará con pulsar sobre "**Sync Now**".

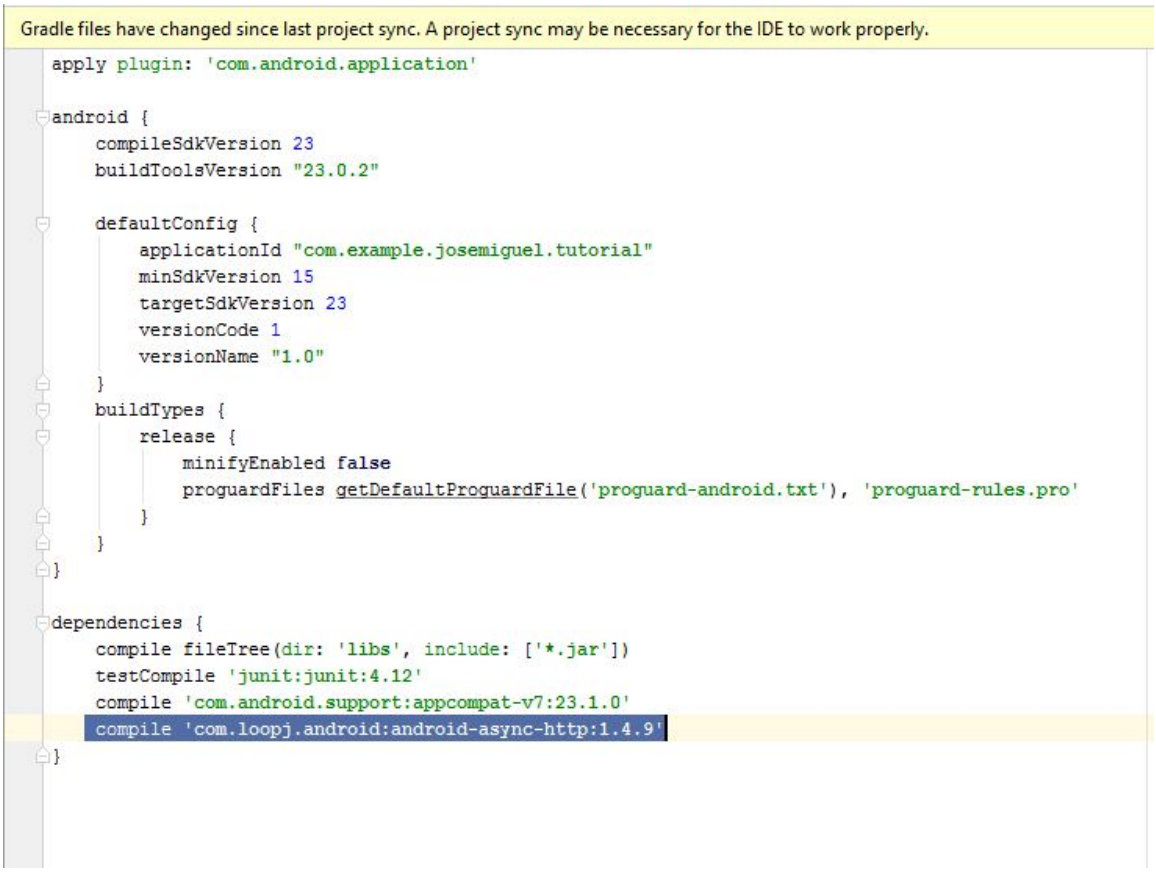

Figura 2: Dependencia android-async-http:1.4.9

- Abrimos el archivo **Manifest.xml**.

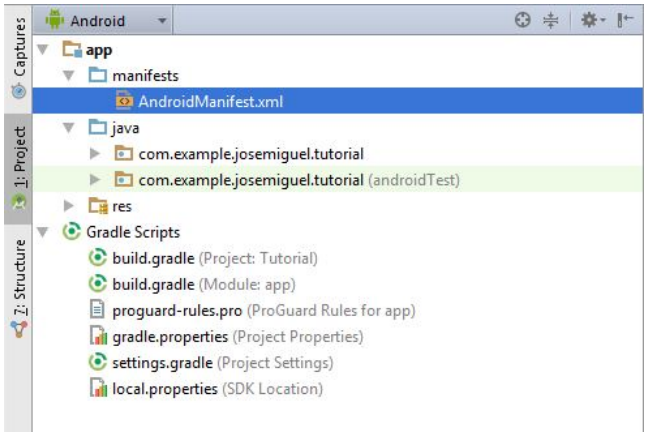

Figura 3: Ubicación del archivo Manifest.xml

- Debemos habilitar los permisos para el uso de internet, con lo que incluiremos la siguiente línea.

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.josemiguel.tutorial" >

    <uses-permission android:name="android.permission.INTERNET" />


    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Tutorial"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 4: Habilitando los permisos de Acceso a Internet.

- Una vez hecho esto, para utilizar **Asynchronous HTTP** solo tendríamos que importarlo utilizando la siguiente línea en la clase Java donde se desee utilizar:

```
import com.loopj.android.http.*;
```

3. Ejemplos de Utilización

3.1. GET

- Ejemplo más básico de GET.

```
public void getprof() {
       AsyncHttpClient client = new AsyncHttpClient();
       client.get("URL", new AsyncHttpResponseHandler() {
           @Override
           public void onSuccess(int statusCode, Header[] headers,
              byte[] responseBody) {
               //Codigo a ejecutar en caso de exito.
           }

           @Override
           public void onFailure(int statusCode, Header[] headers,
              byte[] responseBody, Throwable error) {
               //Codigo a ejecutar en caso de fallo.
           }
       });
   }
```

- La función "**onSuccess**" se ejecutará solo si la llamada al servicio es correcta. En caso contrario, se ejecutará la función "**onFailure**". El resultado de la llamada es devuelto mediante el atributo **responseBody**.

- El entero "statusCode" devolverá el código generado por la operación (GET, POST, PUT...).

- La cadena **URL** representa la dirección del servicio a ejecutar.

3.2. POST

3.2.1. Básico

- Ejemplo más básico de POST.

```
public void EjemploPost(String Parametros){
        AsyncHttpClient client = new AsyncHttpClient();
        client.post("URL" + Parametros, new AsyncHttpResponseHandler()
           {
            @Override
            public void onSuccess(int statusCode, Header[] headers,
               byte[] responseBody) {
                //Codigo a ejecutar en caso de exito.
            }
            @Override
            public void onFailure(int statusCode, Header[] headers,
               byte[] responseBody, Throwable error) {
                //Codigo a ejecutar en caso de error.
            }
        });
    }
```

- Las funciones "**onSuccess**" y "**onFailure**" funcionan exactamente igual que en el GET.
- La cadena "**Parámetro**" simboliza una cadena de caracteres que se pasará al Servicio Web REST mediante el *Path*.

3.2.2. Pasando parámetros de un tipo

- A continuación, se incluye un código de ejemplo de un POST que requiere parámetros de algún tipo concreto, como por ejemplo:
  - **text/plain**
  - **application/json**
  - **application/xml**
- La cadena de caracteres "Tipo" representará este tipo dicho anteriormente.

```
public void EjemploPost1(String parametro){
      AsyncHttpClient client = new AsyncHttpClient();
      String tipo = "Tipo concreto para el POST";
      StringEntity entity = null;
      try {
          entity = new StringEntity(parametro);
      } catch (UnsupportedEncodingException e) {
          e.printStackTrace();
      }
      client.post(this.getApplicationContext(), "URL", entity, tipo, new
          AsyncHttpResponseHandler() {
          @Override
          public void onSuccess(int statusCode, Header[] headers, byte[]
              responseBody) {
              //Codigo a ejecutar en caso de exito.
          }

          @Override
          public void onFailure(int statusCode, Header[] headers, byte[]
              responseBody, Throwable error) {
              //Codigo a ejecutar en caso de error.
          }
      });
  }
```

Referencias

[1] Foro de preguntas y respuestas, Stackoverflow *http://stackoverflow.com/*
Fecha de último acceso: 30 de Noviembre de 2015

[2] Android Asynchronous Http Client *http://loopj.com/android-async-http/*
Fecha de último acceso: 30 de Noviembre de 2015